2015

Authoring Example-based Tutors for Procedural Tasks

Stephen B. Blessing
University of Tampa

Vincent Aleven
Carnegie Mellon University

Stephen B. Gilbert
Iowa State University, gilbert@iastate.edu

Neil T. Heffernan
Worcester Polytechnic Institute

Noboru Matsuda
Carnegie Mellon University

See next page for additional authors

CHAPTER 6 Authoring Example-based Tutors for Procedural Tasks

Stephen B. Blessing[1], Vincent Aleven[2], Stephen B. Gilbert[3], Neil T. Heffernan[4], Noboru Matsuda[2], Antonija Mitrovic[5]
[1] University of Tampa; [2] Carnegie Mellon University; [3] Iowa State University;
[4] Worcester Polytechnic Institute; [5] University of Canterbury

Introduction

Researchers who have worked on authoring systems for intelligent tutoring systems (ITSs) have examined how examples may form the basis for authoring. In this chapter, we describe several such systems, consider their commonalities and differences, and reflect on the merit of such an approach. It is not surprising perhaps that several tutor developers have explored how examples can be used in the authoring process. In a broader context, educators and researchers have long known the power of examples in learning new material. Students can gather much information by poring over a worked example, applying what they learn to novel problems. Often these worked examples prove more powerful than direct instruction in the domain. For example, Reed and Bolstad (1991) found that students learning solely by worked examples exhibited much greater learning than those learning instruction based on procedures. By extension then, since tutor authoring can be considered to be teaching a *tabula rasa* tutor, tutor authoring by use of examples may be as powerful as directly programming the instruction, while being easier to do.

Several researchers have considered how examples may assist programmers in a more general sense (e.g., Nardi, 1993; Lieberman, 2001). This approach, referred to as "programming by example" or "programming by demonstration," generally involves the author programmer demonstrating the procedure in the context of a specific example and then the system abstracting the general rules of the procedure on the basis of machine learning or other artificial intelligence (AI) techniques. The balance in such systems is between its ease of use versus its expressivity. A system may be easy to use, but lack expressive power and thus generality. At the other extreme (e.g., a general-purpose programming language), a system can be very expressive and thus generalizes to new situations readily, but lacks ease of learning. Of course, as an author gets more used to a tool, regardless of initial complexity, the tool becomes easier. The balance between ease of use and expressivity lies with tutor authoring tools as much as it does in the more general case of programming by example.

Some researchers who build authoring systems for ITSs have leveraged this general approach, using examples as a major input method for the ITS. Five such systems are discussed here: Authoring Software Platform for Intelligent Resources in Education (ASPIRE), ASSISTments, Cognitive Tutor Authoring Tools (CTAT), SimStudent, and the Extensible Problem-Solving Tutor (xPST). All of these systems use examples in at least some important aspect of tutor creation. A main goal in using examples is to ease the authoring burden, to both speed up the authoring of ITSs and enable authoring for a wider variety of people. All five systems build tutors for procedural-type tasks, where each step of the task is reasonably well defined and student answers tend to be easily checked. The tutors built by these systems have been deployed in a wide variety of such tasks (e.g., math, chemistry, genetics, statistics, and manufacturing, to name a few). However, some of the systems can also tutor on non-procedural tasks (e.g., ASPIRE). The type of tutoring interaction mediated by these tutors is typically in the pattern of constraint-based and model-tracing tutors. That is, each student step is checked for correctness, with help and just-in-time messages available.

A short description of each of these five systems follows. After these discussions, the general implications for such an example-based method for tutor creation conclude the chapter.

The Authoring Systems

ASSISTments

ASSISTments is a web-based tutoring system started from work on CTAT (discussed below), and developed at both Carnegie Mellon University (CMU) and Worcester Polytechnic Institute (WPI). It is a platform, hosted at WPI, which allows sharing of content between teachers. The platform is domain neutral. ASSISTments gives students problems, and there are content libraries for many disparate subjects including mathematics, statistics, inquiry-based science, foreign language, and reading, but 90% of the content is in mathematics. Each item, or ASSISTment, consists of a main problem and the associated scaffolding questions, hints, and buggy messages.

Early work on this system (circa 2004) required programmers to build content, but soon this was untenable, so a graphical user interface (GUI)-based authoring tool was developed to enable other people, such as teachers and other researchers, to create content in quantity. Figure 1 shows the tutor and authoring screens for the same problem (Razzaq et al., 2009). Somewhere around 2011,the total amount of content created by non-WPI personnel began to outnumber that created by WPI personnel.

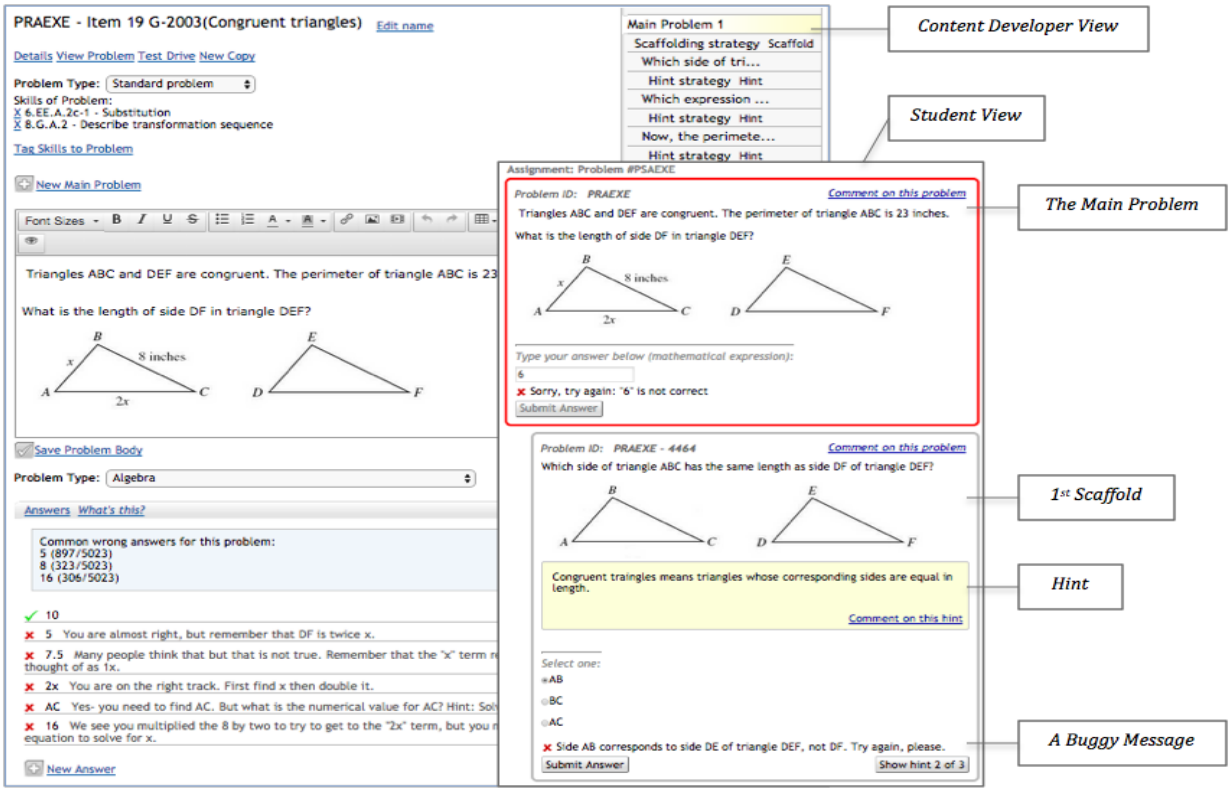

Figure 1. ASSISTments interface.

This is possible because we created an authoring tool that makes it easy to build, test, and deploy items, as well as for teachers to get reports. We have a gentle slope for authors in that they can use our

QuickBuilder to just type in a set of questions and associated answers. In that sense, they have created a simple quiz, where the one hint given would just tell them the answer. For those that want to add further hints to the questions, that step is easy and is part of the QuickBuilder. If they want to create scaffolding questions or feedback messages for common wrong answers, they have to invoke the ASSISTment Builder, requiring a steeper learning curve. While there is a steeper learning curve, we have shown that going through the work of creating scaffolding questions can be very helpful for the lower knowledge students (Razzaq & Heffernan, 2009), but that does not mean that everyone creating content in ASSISTments needs to create both a scaffolding version and a hint version.

This gives teachers the opportunity to create problems specific to their school, for differentiated instruction, or to work with their textbook. All content created by any user can he viewed by, but not edited by, any other user that has the problem number. This makes sharing easy and prevents teachers from having to worry that their content could get "graffiti" on it.

We are exploring a new way of adding content with teachers in Maine. The teacher types in something like the following, "Do #7 from Page 327," so the students have to open their textbook to page 327 to see the seventh question on that page (in doing it this way, the teachers are not violating the copyright of the publisher by duplicating the problem). Teachers can elect for students to receive correctness only feedback or additional tutoring on the homework. The content created around these texts is driven by the teachers and can be shared by anyone using that book. Inspired by Ostrow and Heffernan (2014) that showed video hint messages were more effective that a text version that used the same words, we funded seven teachers to make video hint messages, posted on YouTube or SchoolTube. We are just starting a study to examine the effectiveness of this.

The variabilization feature of the ASSISTments builder allows an author to design one problem and then have many problems created that assess the same skill. This was key to our getting our Skill Builders running. Skill Builders are problem sets that allow a student to keep doing problems until they reach the proficiency threshold, which by default its three correct in a row but can be set by the author. Any teacher that wants to change that simply makes their own copy and changes it.

Figure 2 shows the interface for a variablized assistment. Authors can variablize the hint messages, scaffolding questions, and feedback messages. Authors have to write tiny programs of interconnected variables, which do things like randomly changing the numbers used in the problems. Skill builders are much harder to create, and only a few teachers do this themselves, but WPI has created several hundred for topics from 4th to 10th grade mathematics. Well over half of the teachers use our skill builders.

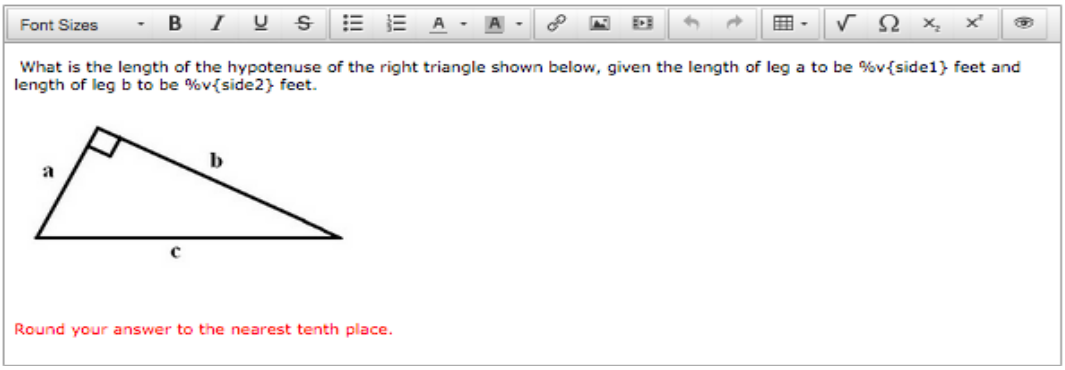

Figure 2. This is a variablized problem on the Pythagorean Theorem.

The authoring tool for ASSISTments has a gentle usability slope. Many teachers start using ASSISTments by first using content WPI created, but most of them soon use the extensibility of the tool to write their own questions. Most of these questions will be what we call "naked," or the lacking of scaffolding hints, as that takes more time to create. We do have some authors that have used the tool to create large libraries of content. For instance, one teacher successfully made hundreds of Advanced Placement (AP) statistics questions with extensive hints.

CTAT

Examples are used extensively in CTAT, a widely used suite of authoring tools (Aleven, McLaren, Sewall & Koedinger, 2009; Aleven, Sewall, McLaren & Koedinger, 2006; Koedinger, Aleven, Heffernan, McLaren & Hockenberry, 2004). CTAT supports the development of tutors that provide individualized, step-by-step guidance during complex problem solving. These tutors provide ample assistance within a problem, such as feedback on the steps, next-step hints, and error feedback messages. They also support individualized problem selection to help each individual student achieve mastery of all targeted knowledge components. Therefore, these tutors support most of the tutoring behaviors identified by VanLehn (2006) as characteristic of ITSs. Over the years, many tutors have been built with CTAT in a very wide range of domains (Aleven et al., 2009; under review). Many of these tutors have been shown to be effective in helping students learn in actual classrooms.

CTAT supports the development of two kinds of tutors: example-tracing tutors, which use generalized examples of problem-solving behavior as their central representation of domain knowledge, and model-tracing tutors (or Cognitive Tutors), which use a rule-based cognitive model for this purpose (Aleven, 2010; Aleven, McLaren, Sewall & Koedinger, 2006). Example-tracing tutors are an innovation that originated with CTAT; this tutoring technology was developed as part of developing CTAT; cognitive tutors on the other hand have a long history that pre-dates CTAT (e.g., Aleven & Koedinger, 2007; Anderson, Corbett, Koedinger & Pelletier, 1995; Koedinger, Anderson, Hadley & Martk, 1997). These two types of tutors support the same set of tutoring behaviors. The main difference is that rule-based cognitive tutors are more practical when a problem can be solved in many different ways (Waalkens, Aleven & Taatgen, 2013). CTAT supports three different approaches to authoring (Figure 3). Example-tracing tutors are built with a variety of end-user programming techniques, including building an interface through drag-and-drop and then programming by demonstration within that interface, where the author's actions are recorded as paths in a behavior graph (Figure 4; the behavior graph is on the right). Rule-based tutors on the other hand can be built in CTAT either through rule-based cognitive modeling, a form of AI programming (Aleven, 2010) or through programming by automated rule induction by a module called SimStudent, which is described in the next section.

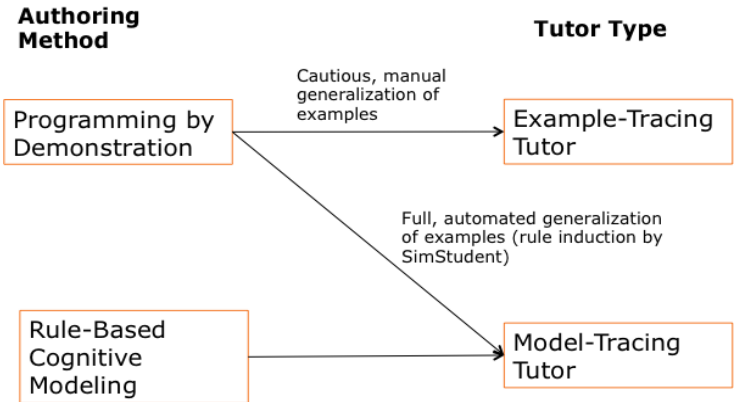

Figure 3. Tutor types and ways of authoring in CTAT

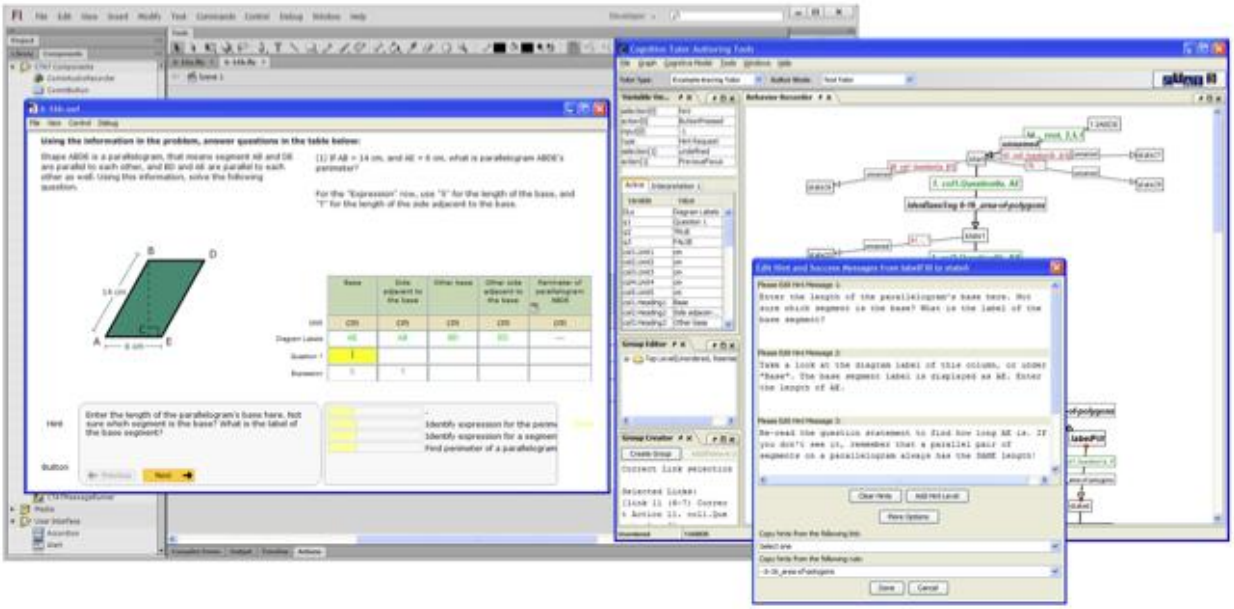

Figure 4. Author using CTAT (right) and Flash (left) to create an example-tracing tutor.

Examples figure prominently in each of these three authoring approaches. These examples take the form of behavior graphs, which capture correct and incorrect problem-solving behavior for the problems that the tutor will help students solve. A behavior graph may have multiple paths, each capturing a different way of solving the problem. Put differently, a behavior graph represents the solution space of a problem. Behavior graphs go at least as far back as Newell and Simon's (1972) classic book *Human Problem Solving*, a foundational work in cognitive science. An author can easily create behavior graphs using CTAT, by demonstrating how to solve problems in the tutor interface. A tool called the Behavior Recorder records the steps in a graph. CTAT also offers tools with which an author can *generalize* a behavior graph, expanding the range of problem-solving behavior that it represents.

Examples serve many different purposes in CTAT. In all three of CTAT's approaches to tutor authoring, examples (i.e., behavior graphs) function as a tool for cognitive task analysis. They help an author map out the solution space of the problems for which tutoring is to be provided, think about different ways a problem might be solved, and develop hypotheses about the particular knowledge components needed and how these components might transfer across steps. In addition, behavior graphs serve various separate functions in each of the authoring approaches. First, in example-tracing tutors generalized examples are the tutor's domain knowledge. The author generalizes the examples in various ways to indicate the range of student behaviors that the tutor will deem correct, so the tutor can be appropriately flexible in recognizing correct student behavior (Aleven, McLaren, Sewall & Koedinger, 2009). Also, in the common authoring scenario that many problems of the same type are needed, an author can turn a behavior graph into a template and create a table with specific values for each problem. Second, in building rule-based cognitive tutors by hand, the examples help in testing and debugging. They help navigate a problem's solution space (e.g., authors can jump to any problem-solving state captured in the graph, which is useful when developing a model from scratch), they serve as semi-automated test cases, and they can be used for regression testing (i.e., making sure that later changes do not introduce bugs). Lastly, in SimStudent, author-demonstrated examples are used to automatically induce production rules that capture the tutor's problem-solving behavior (more detail on this process can be found in the SimStudent section below).

As mentioned, example-tracing tutors use generalized examples (behavior graphs) to flexibly interpret student problem-solving behavior. The tutor checks whether the student follows a path in the graph. Once the student commits to a path, by executing one or more steps on that path, the example-tracer will insist that the student finishes that path, that is, that all subsequent actions are all on at least one path through the graph. Students are not allowed to backtrack and try an alternative problem-solving strategy within the given problem in order to keep them moving forward. Within this basic approach, CTAT's example tracer is very flexible in how it matches a student's problem-solving steps against a behavior graph. First, the example tracer can handle ambiguity regarding which path the student is on and when the steps that the student has entered so far are consistent with multiple paths in a graph. In such situations, the example tracer will maintain multiple alternative interpretations of student behavior until subsequent student steps rule out one or more interpretations. The example tracer also can deal with variations in the order of steps. That is, the student does not need to strictly follow the order in which the steps appear in the graph. An author can specify which parts of a behavior graph require a strict order and which steps can be done in any order. Even better, an author can create a hierarchy of nested groups of unordered and ordered steps. Further, steps can be marked as optional or repeatable. The example tracer can also deal with variations of the steps themselves. An author has a number of ways to specify a range of possibilities for a particular steps, including range matches, wildcard matches, regular expressions, as well as an extensible formula language for specifying calculations and how a step depends on other steps. Thus, in CTAT example-tracing tutors, a behavior graph can stand for a wide range of behavior well beyond exactly the steps in the graph in exactly the order they appear in the graph. Authors have many tools that enable them to specify how far to generalize. When an author wants to make behavior graphs for many different but isomorphic problems, CTAT provides a "Mass Production" approach in which an author creates a behavior graph with variables for the problem-specific values and then, in Excel, creates a table with problem-specific values for a range of problems. They can then generate specific instances of the template in a merge step. This template-based process greatly facilitates the creation of a series of isomorphic problems, as are typically needed in tutor development.

Our experience over the years, both as developers of example-tracing tutors and consultants assisting others in developing example-tracing tutors, indicates that this type of tutor is useful and effective in a range of domains. It also indicates that the example-tracing technology implemented in CTAT routinely withstands the rigors of actual classroom use. Examples of example-tracing tutors recently built with CTAT and used in actual classrooms are Mathtutor (Aleven, McLaren & Sewall, 2009), the Genetics Tutor (Corbett, Kauffman, MacLaren, Wagner & Jones, 2010), the Fractions Tutor (Rau, Aleven & Rummel, 2015; Rau, Aleven, Rummel & Pardos, 2014), a version of the Fractions Tutor for collaborative learning (Olsen, Belenky, Aleven & Rummel, 2014; Olsen, Belenky, Aleven, Rummel, Sewall & Ringenberg, 2014), a fractions tutor that provides grounded feedback (Stampfer & Koedinger, 2013), the Stoichiometry Tutor (McLaren, DeLeeuw & Mayer, 2011a; 2011b), AdaptErrEx (Adams et al., 2014; McLaren et al., 2012), an English article tutor (Wylie, Sheng, Mitamura & Koedinger, 2011), Lynnette, a tutor for equation solving (Long & Aleven, 2013; Waalkens et al., 2013), and a tutor for guided invention activities (Roll, Holmes, Day & Bonn, 2012). We have also seen, in courses, workshops, and summer schools that we have taught, that learning to build example-tracing tutors with CTAT can be done in a relatively short amount of time. Generally, it does not take more than a couple of hours to get started, a day to understand basic functionality, and a couple more days to grasp the full range of functionality that this tutoring technology offers. This is a much lower learning curve than that for learning to build cognitive tutors with CTAT. Authoring and debugging a rule-based cognitive models is a more complex task that requires AI programming. Example-tracing tutors on the other hand do not require any programming. In our past publication (Aleven, McLaren, Sewall & Koedinger, 2009), we estimated, based on data from projects in which example-tracing tutors were built and used in real educational settings (i.e., not just prototypes) that example-tracing tutors make tutor development 4–8 times more cost-effective: they can be developed faster and do not require expertise in AI programming. Echoing a theme that runs throughout the chapter, we emphasize that building a good tutor requires more than being

facile with authoring tools; for example, it also requires careful cognitive task analysis to understand student thinking and students' difficulties in the given task domain.

In sum, the CTAT experience indicates that the use of examples, in the form of behavior graphs that capture the solution space of a problem, is key to offering easy-to-learn, non-programmer options to ITS authoring. Thinking in terms of examples and concrete scenarios is helpful for authors. So is avoiding actual coding, made possible by the use of examples. The experience indicates also that the same representation of problem-solving examples, namely, behavior graphs, can serve many different purposes. This versatility derives from the fact that behavior graphs are a general representation of problem-solving processes. As such, they may be useful in a range of ITS authoring tools, not just CTAT, since many ITSs deal with complex problem-solving activities.

SimStudent

SimStudent is a machine-learning agent that inductively learns problem-solving skills (Li, Matsuda, Cohen & Koedinger, 2015; Matsuda, Cohen & Koedinger, 2005). At an implementation level, SimStudent acts as a pedagogical agent that can be interactively tutored. SimStudent is a realization of programming by demonstration (Cypher, 1993; Lau & Weld, 1998) in the form of inductive logic programming (Muggleton & de Raedt, 1994). SimStudent learns domain principles (i.e., how to solve problems) by specializing and generalizing positive and negative examples on how to apply, and not to apply, particular skills to solve problems.

At a theory level, SimStudent is a computational model of learning that explains both domain-general and domain-specific theories of learning. As for the domain-general theory of learning, SimStudent models two learning strategies: learning from examples and learning by doing (Matsuda, Cohen, Sewall, Lacerda & Koedinger, 2008). *Learning from examples* is a model of passive learning in which SimStudent is given a set of worked-out examples and it silently generalizes solution steps from these examples. There is no interaction between the "tutor" and SimStudent during learning from examples, except that tutor provides examples to SimStudent. *Learning by doing*, on the other hand, is a model of interactive, tutored-problem solving (i.e., cognitive tutoring) in which SimStudent is given a sequence of problems and asked to solve them. In this context, there must be a "tutor" (i.e., author) who provides tutoring scaffolding (i.e., feedback and hints) to SimStudent. That is, the "tutor" provides immediate flagged feedback (i.e., correct or incorrect) for each of the steps that SimStudent performs. SimStudent may get stuck in the middle of a solution and ask the "tutor" for help on what to do next. The "tutor" responds to SimStudent's inquiry by demonstrating the exact next step.

As for the domain-specific theory of learning, SimStudent can be used as a tool for student modeling to advance a cognitive theory of learning skills to solve problems for a particular domain task. Using the SimStudent technology, researchers can conduct simulation studies with tightly controlled variables. For example, to understand why students make commonly observed errors when they learn how to solve algebraic linear equations, we conducted a simulation study. An example of a common error is to subtract 4 from both sides of $2x–4=5$. We hypothesized that students learn skills incorrectly due to incorrect induction. We also hypothesized that incorrect induction might more likely occur when students carry out induction based on weak background knowledge that, by definition, is perceptually grounded and therefore lacks connection to domain principles. An example of such weak background knowledge is to perceive "3" in $5x+3=7$ as *a last number* on the left-hand side of the equation, instead of perceiving '+3' as a *last term*. To test these hypotheses, we controlled SimStudent's background knowledge by replacing some of the background knowledge (e.g., the knowledge to recognize the last term) with weak perceptually grounded knowledge (e.g., the knowledge to recognize the last number). We trained two versions of SimStudent (one with normal background knowledge and the other one with weak

background knowledge) and compared their learning with students' learning. The result showed that only SimStudent with weak background knowledge made the same errors that students commonly make (Matsuda, Lee, Cohen & Koedinger, 2009).

So far, we have demonstrated that SimStudent can be used to advance educational studies for three major problems: (1) intelligent authoring, (2) student modeling, and (3) teachable agent. For intelligent authoring, SimStudent functions as an intelligent plug-in component for CTAT (Aleven, McLaren, Sewall & Koedinger, 2006; Aleven, McLaren, Sewall & Koedinger, 2009) that allows authors to create a cognitive model (i.e., a domain expert model) by tutoring SimStudent on how to solve problems. The intelligent authoring project was started as an extension of prior attempts (Jarvis, Nuzzo-Jones & Heffernan, 2004; Koedinger, Aleven & Heffernan, 2003; Koedinger, Aleven, Heffernan, McLaren & Hockenberry, 2004).

In the context of intelligent authoring, the author first creates a tutoring interface using CTAT, and then "tutors" SimStudent using the tutoring interface (Figure 5). There are two authoring strategies, *authoring by tutoring* and *authoring by demonstration*, and each corresponds to two learning strategies mentioned above, i.e., learning by doing and learning from worked-out examples, respectively. We have showed that when the quality of a cognitive model is measured as the accuracy of solution steps suggested by the cognitive model, authoring by tutoring generates a better cognitive model than authoring by demonstration (Matsuda, Cohen & Koedinger, 2015). It is only authoring by tutoring that provides negative examples, which by definition tell SimStudent when not to apply overly general productions, and negative examples have the significant role in inductively generating a better quality cognitive model.

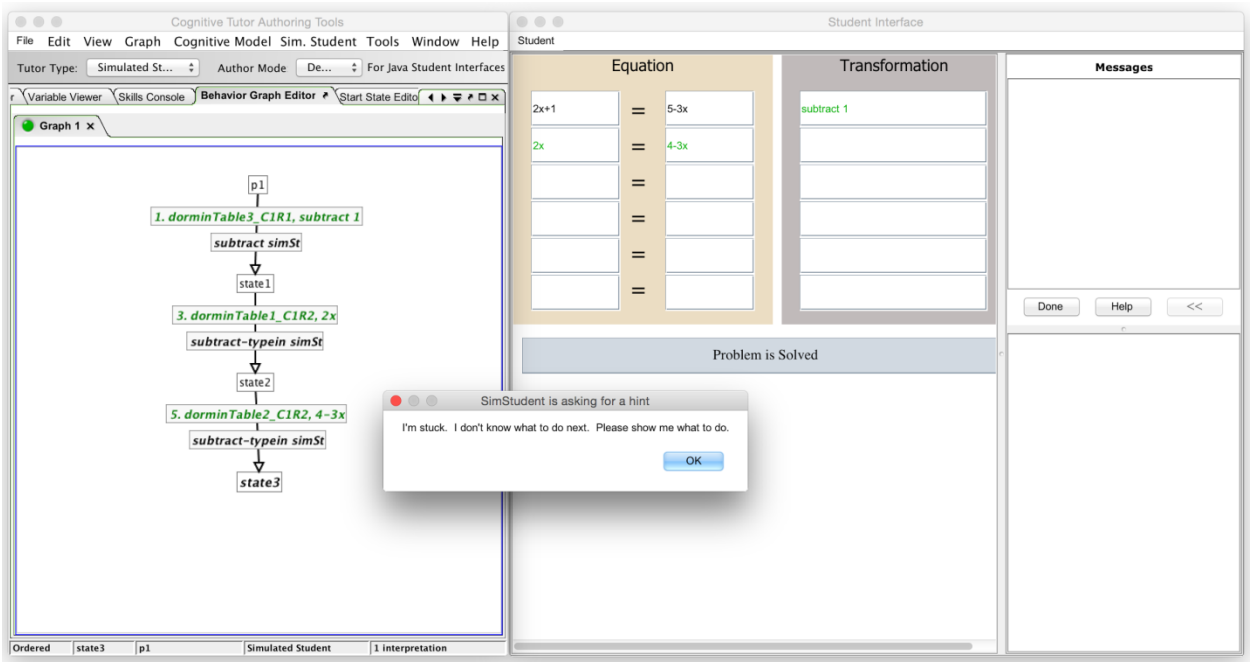

Figure 5. Authoring using SimStudent with the assistance of CTAT

SimStudent also functions as a teachable agent in an online learning environment in which students learn skills to solve problems by interactively teaching SimStudent. The online learning environment is called the Artificial Peer Learning environment Using SimStudent (APLUS). APLUS and a cognitive tutor share underlying technologies. In fact, APLUS consists of (1) the tutoring interface on which a student tutors

SimStudent; (2) a cognitive tutor in the form of the *meta-tutor* that provides scaffolding for the student on how to teach SimStudent and how to solve problems; and (3) a teachable agent (SimStudent), with its avatar representation. The combination of CTAT and SimStudent allows users to build APLUS for their own domains. In this context, SimStudent plays a dual role: (1) a tool to create a cognitive model for the embedded meta-tutor and (2) a teachable agent.

Examples, in the context of the interaction with SimStudent, are major input for SimStudent to induce a cognitive model. SimStudent learns procedural skills to solve target problems either from learning by doing or learning from worked-examples. SimStudent generalizes provided examples (both positive and negative) and generates a set of productions that each represents a procedural skill. The set of productions become a cognitive model that can be used for cognitive tutoring in the form of a cognitive tutor or a meta-tutor in APLUS.

An empirical study (Matsuda et al., 2015) showed that to make an expert model for an algebra cognitive tutor, it took a subject matter expert 86 minutes for *authoring by tutoring* SimStudent on 20 problems whereas *authoring by demonstration* with 20 problems took 238 minutes. A more recent study showed that authoring an algebra tutor in SimStudent is 2.5 times faster than example-tracing while maintaining equivalent final model quality (MacLellan, Koedinger & Matsuda, 2014). We are currently conducting a study to validate the quality of production rules. In the study, we actually use a SimStudent-generated cognitive model for an algebra cognitive tutor to model trace real student's solution steps. A preliminary result shows that after tutoring SimStudent on 37 problems, the model tracer correctly model traces 96% of steps that students correctly performed. At the same time, the "accuracy" of detecting a correct step (i.e., the ratio of the correct positive judgement, judging a step as correct, to all positive judgement) was 98%.

ASPIRE

The Intelligent Computer Tutoring Group (ICTG; http://www.ictg.canterbury.ac.nz/) has developed many successful constraint-based tutors in diverse instructional domains (Mitrovic, Martin & Suraweera, 2007; Mitrovic, 2012). Some early comparisons of constraint-based modeling (Ohlsson, 1994) to the model-tracing approach have shown that constraint-based tutors are less time-consuming to develop (Ohlsson & Mitrovic, 2007; Mitrovic, Koedinger & Martin, 2003), but yet require substantial expertise and effort. The estimate of time per constraint for Structured Query Language (SQL)-Tutor, the first and biggest constraint-based modeling (CBM) tutor developed (Mitrovic, 1998), was 1 hour per constraint, with the same person acting as the knowledge engineer, domain expert, and software developer. In order to support the development process, ICTG developed an authoring shell, the Web-Enabled Tutor Authoring System (WETAS; Martin & Mitrovic, 2002). Studies with novice ITS authors using WETAS had shown that the authoring time per constraint on average was 2 hours (Suraweera et al., 2009), but the authors still found writing constraints challenging.

ASPIRE (http://aspire.cosc.canterbury.ac.nz/) is a general authoring and deployment system for constraints-based tutors. It assists in the process of composing domain models for constraint-based tutors and automatically serves tutoring systems on the web. ASPIRE guides the author through building the domain model, automating some of the tasks involved, and seamlessly deploys the resulting domain model to produce a fully functional web-based ITS.

The authoring process in ASPIRE consists of eight phases. Initially, the author specifies general features of the chosen instructional domain, such as whether or not the task is procedural. For procedural tasks, the author describes the problem-solving steps. This is not a trivial activity, as the author needs to decide on

the approach to teaching the task. The author also needs to decide on how to structure the student interface and whether the steps will be presented on the same page or on multiple pages.

The author then develops the domain ontology, containing the concepts relevant to the instructional task. The purpose of the domain ontology is to focus the author on important domain concepts; ASPIRE does not require a complete ontology, but only those domain concepts students need to interact with in order to solve problems in the chosen area. The ontology specifies the hierarchical structure of the domain in terms of sub- and super-concepts. Each concept might have a number of properties and may be related to other domain concepts. The author can define restrictions on properties and relationships, such as the minimum and maximum, number of values, types of values, etc. The ontology editor does not offer a way of specifying restrictions on different properties attached to a given concept, such as the number of years of work experience should be less than the person's age. It also does not contain functionality to specify restrictions on properties from different concepts, such as the salary of the manager has to be higher than the salaries of employees for whom they are responsible. However, these restrictions are not an obstacle for generating the constraint set, as ASPIRE generates constraints not only from the ontology, but also from sample problems and their solutions. Figure 6 shows the domain ontology for the thermodynamics tutor, which is defined as a procedural task. In this tutor, the student needs to develop a diagram first and later compute unknowns using a set of formulas.

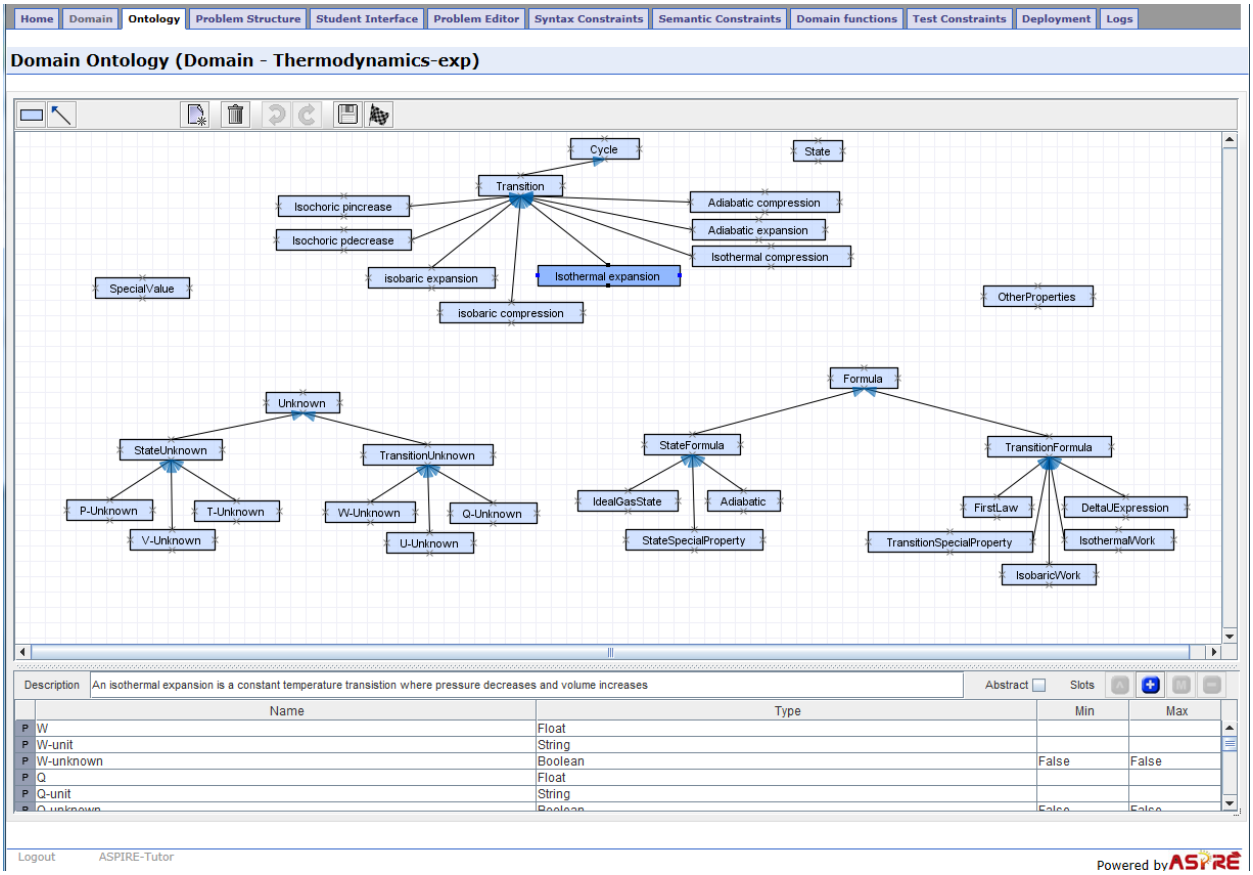

Figure 6: The ontology of Thermo-Tutor

In the third phase, the author defines the problem structure and the general structure of solutions, expressed in terms of concepts from the ontology. The author specifies the types of components to show on the student interface and the number of components (e.g., a component may be optional or can have

multiple instances). On the basis of the information provided by the author in the previous phases, ASPIRE then generates a default, text-based student interface, which can be replaced with a Java applet. ASPIRE also provides a remote procedure call interface, allowing for sophisticated student interfaces to be built, such as an Augmented Reality interface (Westerfield, Mitrovic & Billinghurst, 2013). Figure 7 shows the Java applet allowing students to solve problems in Thermo-Tutor (Mitrovic et al., 2011).

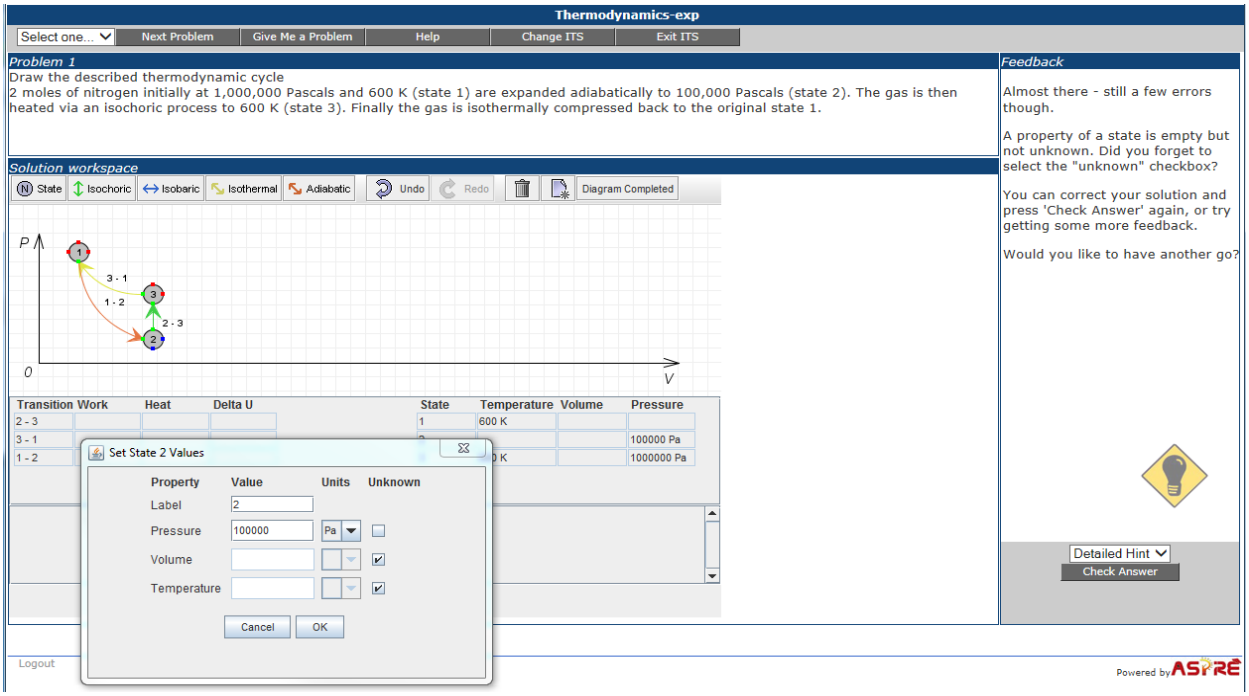

Figure 7: A screenshot from Thermo-Tutor showing the applet

In the fifth phase, the author adds sample problems and their correct solutions using the problem solution interface. ASPIRE does not require the author to specify incorrect solutions. The interface enforces that the solutions to adhere to the structure defined in the previous step. The author is encouraged to provide multiple solutions for each problem, demonstrating different ways of solving it. In domains where there are multiple solutions per problem, the author should enter all practicable alternative solutions. The solution editor reduces the amount of effort required to do this by allowing the author to transform a copy of the first solution into the desired alternative. This feature significantly reduces the author's workload because alternative solutions often have a high degree of similarity.

ASPIRE then generates syntax constraints by analyzing the ontology and the solution structure. The syntax constraint generation algorithm extracts all useful syntactic information from the ontology and translates it into constraints. Syntax constraints are generated by analyzing relationships between concepts and concept properties specified in the ontology (Suraweera, Mitrovic & Martin, 2010). An additional set of constraints is also generated for procedural tasks, which ensure the student performs the problem-solving steps in the correct order (also called *path constraints*).

Semantic constraints check that the student's solution has the desired meaning (i.e., it answers the question). Constraint-based tutors determine semantic correctness by comparing the student solution to a single correct solution to the problem; however, they are still capable of identifying alternative correct solutions because the constraints are encoded to check for equivalent ways of representing the same semantics (Ohlsson & Mitrovic, 2007; Mitrovic, 2012). ASPIRE generates semantic constraints by analyzing alternative correct solutions for the same problem supplied by the author. ASPIRE analyses the

similarities and differences between two solutions to the same problem. The process of generating constraints is iterated until all pairs of solutions are analyzed. Each new pair of solutions can lead to either generalizing or specializing previously generated constraints. If a newly analyzed pair of solutions violates a previously generated constraint, its satisfaction condition is generalized in order to satisfy the solutions, or the constraint's relevance condition is specialized for the constraint to be irrelevant for the solutions. A detailed discussion of the constraint-generation algorithms is available in (Suraweera, Mitrovic & Martin, 2010).

xPST

When an author uses the xPST system to create a model-tracing style tutor (e.g., Koedinger, Anderson, Hadley & Mark, 1997) for a learner, the author bases the instruction on a particular example. The example needs to already be in existence—xPST does not provide a way to create that example. Rather, that example comes from previously created content or is based on third-party software. This aspect of the system is contained in its name—problem-specific tutor. Very little generalization is done from the example. Broadly speaking, the instruction that the author creates is appropriate only for that one example. While this limits the ability for the instruction to be applied in multiple instances, it allows for a more streamlined and simplistic authoring process, opening up the possibility of authoring tutors to a wider variety of people, e.g., those who do not possess programming skills.

To quickly explain the first word in the xPST name, extensible, that ability comes from two different aspects. First, xPST can be extended in terms of the types of learner answers it can check. xPST's architecture compartmentalizes these "checktypes," and it is easy for a programmer to add additional ones and make them available to xPST authors. Second, and more importantly, xPST can be extended in terms of the interfaces on which it can provide tutoring. Like other ITSs (such as seen in CTAT, or see Blessing, Gilbert, Ourada, and Ritter, 2009; Ritter & Koedinger, 1996), xPST's architecture makes a clear separation between the learner's interface and the tutoring engine. The architecture contains a TutorLink module that mediates the communication between these two parts of the system. The learner's interface can in theory be any existing piece of software, as long as a TutorLink module can translate the actions of the learner in the interface into what xPST understands, and then the module needs to communicate the tutoring feedback back to the learner's interface (e.g., a help message or an indication if an answer is right or wrong). More information concerning this type of communication can be found elsewhere (Gilbert, Blessing & Blankenship, 2009).

Allowing the learner interface to be existing software, given the proper TutorLink module, opens up many possibilities in terms of what to provide tutoring on and how that tutoring manifests itself. We have written TutorLink modules for Microsoft .NET programs, the Torque 3-D game engine, and the Firefox web browser. Regardless of the interface, the authoring interaction is similar: a specific scenario is created within the context of the interface and instruction on completing that scenario is authored in xPST. To explain how examples are used to create tutoring in xPST, we illustrate the process using the Firefox web browser as the interface. In this case, the TutorLink module operates as a Firefox plug-in. This allows any webpage to contain potentially tutorable content, where the student is provided with model-tracing style feedback. In one project, we had authors, which included non-programmer undergraduates, use a drag-and-drop form creation tool to easily create custom homework problems for a statistics tutor (Maass & Blessing, 2011). Countless webpages already exist that could be used for instruction. In another project, we used a webpage from the National Institutes of Health (NIH) to create activities involving DNA sequencing.

To provide a specific example, imagine an author wanted to create instruction on how to search using a popular article database, the American Psychological Association's (APA) PsychINFO, to find research

papers, so that students become better at information literacy. The webpage already exists, with all the widgets (the entry boxes, radio buttons, and pull-down menus) in place. The Firefox plug-in allows the author to write a problem scenario (e.g., to find a particular paper using those widgets) that will appear in a sidebar next to the already established page, and then the author writes instruction code that will ensure that the learner uses the page appropriately, providing help when needed, so that the learner finds the correct article. The author does their work on the xPST website (http://xpst.vrac.iastate.edu). This website provides a form to create a new problem, where the instruction to the existing webpage (in this case, http://search.proquest.com/psychinfo/advanced/), the sidebar's problem scenario, and the tutor "code" that contains the right answers and help messages can all be entered. While the code does have some of the trappings of traditional programming, those are kept to a minimum.

Figure 8 shows some of the code that would be used to create this PsychINFO tutor. This code in conjunction with the author-supplied scenario is in essence the example. The existing webpage provides the means by which the learner will work through the example (via the entry boxes and drop-down menus), and what is seen in Figure 8 is the information needed by xPST to provide tutoring. The code has three main sections: Mappings, Sequence, and Feedback. The Mappings map the interface widgets onto the names that the xPST tutor will use. The Firefox plug-in provides the names of the widgets for the author as the author begins to create the scenario. The Sequence is the allowed orderings for how the learner may progress through the problem. The syntax allows for required and optional parts, along with different kinds of branching. The Feedback section is where the author indicates the right answer for a widget, and the help and just-in-time messages that might be displayed for incorrect responses. Once authors have entered in enough code to see results, they can click the "Save and Run" button and immediately see the results of the xPST tutor. Figure 9 shows the tutor running the PsychINFO site with the code shown in Figure 8. This code is specific to this problem scenario, but could easily be copied and modified in order to create a different problem. In such a way an author could quickly create a short 5–6 problem homework set to provide practice to students concerning information literacy.

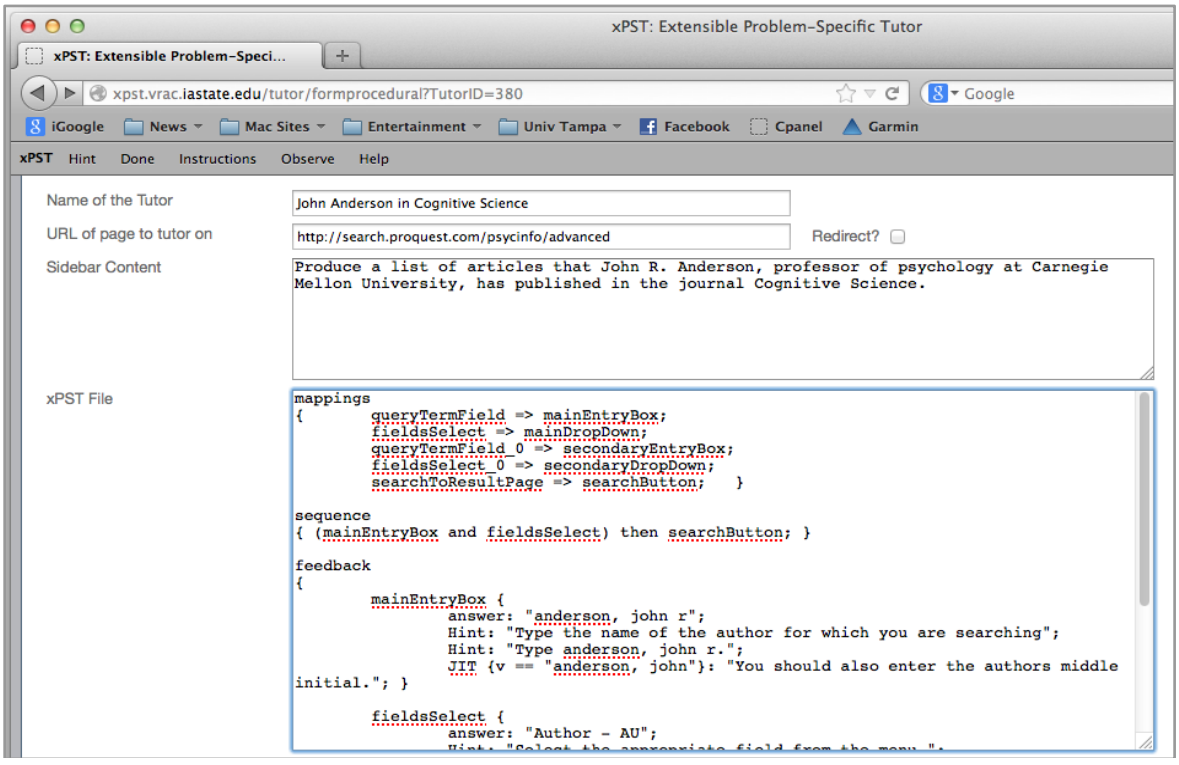

Figure 8. Authoring interface for xPST.

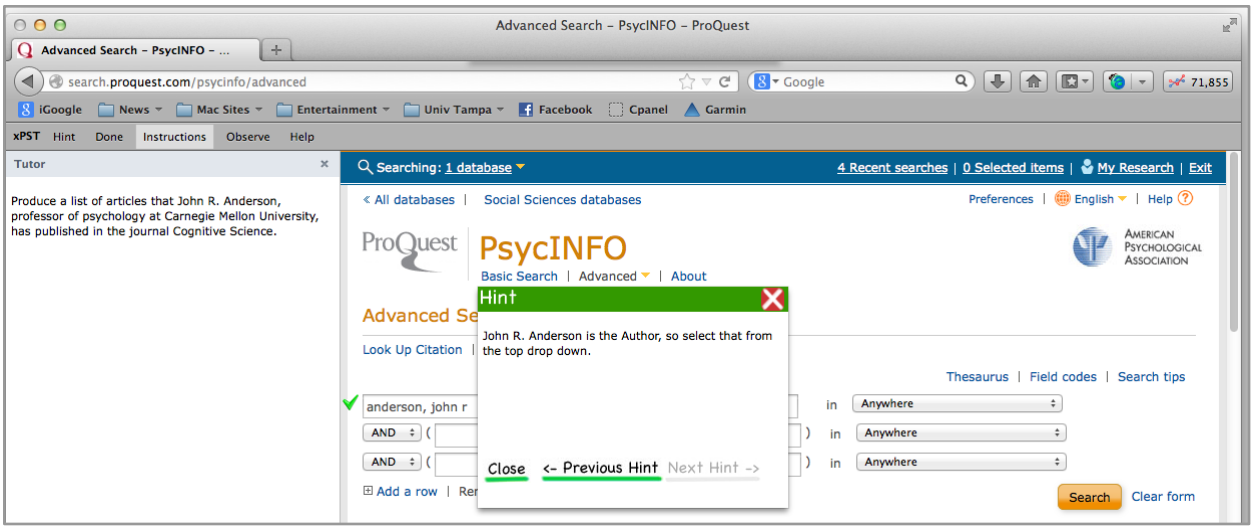

Figure 9. The example-based tutor running on the PsychInfo site.

We have examined the way non-programmers have learned to use xPST (e.g., Blessing, Devasani & Gilbert, 2011). Despite the text-entry method for instruction, non-programmers have successfully used xPST to create new tutors. In Blessing, Devasani, and Gilbert (2011), five such authors spent roughly 30 hours on average learning the system and developing 15 statistics problems apiece. Keeping in mind that all the problems had a similar feel to them, the endpoint was the ability to create one of the problems, which contained about 10 minutes of instruction, in under 45 minutes.

Conclusions

We start our conclusions by comparing the above systems on five dimensions: (1) their heritage, (2) practical concerns such as teacher reporting, (3) the authoring process, (4) how they generalize examples, and (5) their approach to cognitive task analysis. We finish by making recommendation to the Generalized Intelligent Framework for Tutoring (GIFT) architecture based on our observations.

Heritage

Four of these five systems (ASSISTments, CTAT, SimStudent, and xPST) share a common heritage, the ACT Tutors that John Anderson and his colleagues developed over the course of many years (Anderson, Corbett, Koedinger & Pelletier, 1995). The researchers created these tutors to fully test the ACT Theory of cognition, and they covered a few different domains, including several programming languages and many levels of mathematics. The most direct descendant of the ACT Tutors existing today are the commercial tutors produced by Carnegie Learning, Inc., which cover middle and high school math.

Despite this common heritage of the present systems, they were developed independently. Each of us felt that the authoring tools created to support the ACT Tutors (the Tutor Development Kit for the original set of tutors (Anderson & Pelletier, 1991) and the Cognitive Tutor Software Development Kit for Carnegie Learning's tutors (Blessing, Gilbert, Ourada & Ritter, 2009)), while powerful, were not approachable by non-programmers or non-cognitive scientists. We realized that in order for ITSs to be more prevalent, authoring needed to be easier. In our own labs, we developed separate systems that mimicked the behavior of the original ACT Tutors, because that had proved so successful, but without the programming

overhead that prior tools required. As seen in our descriptions above and our discussion here, these systems contain some similarities, but differ in important ways as well.

ASPIRE, the one system that does not have a connection to the ACT Tutors, originated with Ohlsson's work on a theory of learning from performance errors (Ohlsson, 1996). This led to the development of CBM (Mitrovic & Ohlsson, 1999), in which the tutor's knowledge is represented as a set of constraints, as opposed to the production-based representation of the ACT Tutors. In this way, the tutor's knowledge represents boundary points within which the solution lies. Having multiple systems that descend from multiple sources provides credence to the idea that the general technique of programming by demonstration and the use of examples is a useful and powerful one for the creation of ITSs.

Practical Concerns

There are scientific concerns as to what knowledge representations are most valuable to use to reflect how humans think (e.g., Ohlsson's constraints-based theory vs. Anderson's production rules). However, there are also practical concerns. For example, which tools prove easier to use might drive adoption, not necessarily those that produce the most learning. As another somewhat practical concern, some of the authoring methods discussed above may allow authors to more easily add complexity to their content over time. For instance, after assigning a homework question, a teacher may see that an unanticipated common wrong answer occurs, and the system needs to allow the teacher to write a feedback message that addresses that common wrong answer quickly.

While this chapter has focused on author tools for the content, an equally important element has to do with reporting. Some of these tools, such as ASSISTments and CTAT offer very robust ways to report student data. There is a possible tradeoff on the complexity and adaptability of the content, and the ways we report to instructors. We need easy ways that report information to the instructors and content creators. The reports to these classes of people should be focused differently than the types of reports to researchers. For instance, if a researcher has used ASSISTments' tools to create a randomized controlled experiment (see sites.google.com/site/neilheffernanscv/webinar for more information concerning this feature) embedded in a homework, perhaps comparing text hints versus video hints, the reports that the teachers receive should be different than the reports that the researchers receive.

The Authoring Process

The method by which authors create tutors in these systems varies along at least two different, though somewhat related, dimensions: (1) how the instruction is inputted and (2) how much of the process is automated. With regard to how the instruction is inputted, this varies from a method that is more traditional coding as in xPST, to a method that is more graphical in nature, such as CTAT's behavior graph. ASSISTment's QuickBuilder and ASSISTment Builder techniques seem to be a bit of a midpoint between those two methods of input. Devasani, Gilbert, and Blessing (2012) examined the trade-offs between these approaches with novice authors building tutors in both CTAT and xPST within two different domains, statistics and geometry. Relating their findings to Green and Petre's (1996) cognitive dimensions, they argued that the GUI approach has certain advantages, such as eliminating certain types of errors and the fact that visual programming allows for a more direct mapping. A more text-based approach has the advantages of flexibility in terms of how the authoring is completed and the ability to capture larger tutors that contain more intermediate states and solution paths more economically (what Green and Petre termed "diffuseness" and "terseness"). That flexibility may also translate into easier maintenance of those larger tutors.

The systems also differ in how much of the process is automated. This is also related to the amount of generalizability that the systems are able to perform, discussed below. In both ASSISTments and xPST, very little, if anything, is automated. ASPIRE and SimStudent have some degree of automation, in terms of how they induce constraints or productions. This automation eliminates or reduces greatly some of the steps that the author would otherwise have to do in order to input the instruction. CTAT is the middle system here, as it does have some mechanisms available to the author to more automatically created instruction (e.g., using Excel to more quickly create problem sets that all share similar instruction). As in any interface and systems design, these two dimensions play off each other in terms of what advantages they offer the author, between ease-of-use and generalizability.

Generalization of Examples

The discussed authoring technologies are diverse: they help authors create different kinds of domain models that can be used for adaptive tutoring. Some help authors create a collection of questions and answers with knowledge of feedback (ASSISTments, the example-tracing version of CTAT, and xPST), whereas others provide scaffolding to create the domain model either in the form of constraints (ASPIRE) or production rules (the model-tracing version of CTAT and SimStudent).

Some of the discussed approaches rely on the author's ability for programming while providing elaborated scaffolding to facilitate the programming process and ease the author's labor (ASSISTments, CTAT, xPST). In xPST, there is no generalization of examples at all. In CTAT, the author specifies the behavior graphs that includes both correct and incorrect steps, and also provides feedback on steps and hints. Furthermore, the author generalizes examples by adding variables and formulas that express how steps depend on each other or how a given vary, by relaxing ordering constraints, and by marking steps as optional or repeatable. In ASSISSTments, some variabilization is possible. In those two cases, the authoring system does not generalize examples on its own; this task is left to the author.

On the other hand, ASPIRE and SimStudent deploy AI technologies to generate the domain model given appropriate background knowledge. ASPIRE, for example, generates constraints given the domain ontology developed by the author and example solutions. SimStudent uses the given primitive domain skills to generate a cognitive model from a set of positive and negative examples provided by the author. The difference between ASPIRE and SimStudent is not only in the formalism in which domain knowledge is represented, but also in the kind of examples they use. ASPIRE requires the author to specify only the alternative correct solutions for problems, without any feedback or further elaborations on them. SimStudent requires immediate feedback on steps (when inducing production rules in the learning by doing mode) or a set of positive and negative examples.

All five authoring systems discussed in this chapter share a common input for tutor authoring—example solutions. Different techniques are used for different purposes to generalize or specialize the given examples. It must be noted that all these five authoring systems share a fundamentally comparable instructional strategy for procedural tasks, step decomposition (i.e., force students to enter a solution one step at a time). ASPIRE differs from this requirement, as it can also support non-procedural tasks, in which the student can enter the whole solution at once. With the exception of ASPIRE, which provides on-demand feedback, the other authoring systems provide immediate (or semi-immediate) feedback on the correctness of the step performed, and just-in-time hint on what to do next.

Cognitive Task Analysis

Performing a cognitive task analysis (CTA) has been shown to be an effective means of producing quality instruction in a domain (Clark & Estes, 1996). CTA involves elucidating the cognitive structures that

underlie performance in a task. Another aspect of CTA is to describe the development of that knowledge from novice to expert performance. The more ITS authors (or any other designers of instruction) understand about how students learn in the given task domain, what the major hurdles, errors, and misconceptions are, and what prior knowledge students are likely to bring to bear, the better off they are. This holds for designing many, if not all, other forms of instruction, regardless of whether any technology is involved.

The space of cognitive task methods and methodologies is vast (Clark, Feldon, van Merriënboer, Yates & Early, 2007). Some of these techniques have been applied successfully in tutor development (Lovett, 1998; Means & Gott, 1988; Rau, Aleven, Rummel & Rohrbach, 2013). Two techniques that have proven to be particularly useful in ITS development, though not the only ones, are think-aloud protocols and a technique developed by Koedinger called difficulty factors assessment (DFA; Koedinger & Nathan, 2004). DFA is a way of creating a test (with multiple forms and a Latin-Square logic) designed to evaluate the impact on student performance of various hypothesized difficulty factors. Creating these tests is somewhat of an art form, but we may see more data-driven and perhaps crowd-based approaches in the future. Baker, Corbett, and Koedinger (2007) discussed how these two forms of cognitive task analysis can help, in combination with iterative tutor development and testing to detect and understand design flaws in a tutor and create a more effective tutor. Interestingly, in the area of ITS development, manual approaches to CTA are more and more being supplemented by automated or semi-automated approaches, especially in the service of building knowledge component models that accurately predict student learning (Aleven & Koedinger, 2013). CTA is important to ITS development, as it is for other forms of instructional design. The more instruction is designed with a good understanding of where the real learning difficulties lie, the more effective the instruction is going to be. ITSs are no exception. This point was illustrated in the work by Baker et al. (2007) on a tutor for middle school data analysis—CTA helped make a tutor more effective. Outside the realm of ITSs, this point was illustrated in the redesign of an online course for statistics, using CTA, where the redesigned course was dramatically more effective (Lovett, Myers & Thille, 2008).

Given the importance of CTA in instructional design, we should ask to what degree ITS authoring tools support any form of CTA and in what ways they are designed to take advantage of the results of CTA to help construct an effective tutor and perhaps make tutor development more efficient. For example, one function of the behavior graphs used in CTAT is as a CTA tool. The other authoring tools described here make use of CTA in various ways as the author creates a tutor. Although mostly implicit in their design, the authoring systems depend on authors having performed an adequate task decomposition in their initial interface construction, sometimes referred to as subgoal reification (Corbett & Anderson, 1995). Without the author having enabled the learner to make explicit their thought processes as they use the tutor, then attempts at assessing their current state of knowledge or addressing any deficiency will be greatly diminished. Therefore, before beginning the writing of any help or just-in-time messages, it is crucial to have the student's interface support the appropriate tasks needed to be performed by the student.

As mentioned, CTA is supported in CTAT, as the easily recorded behavior graphs. A behavior graph is a map of the solution space for a given problem for which the tutoring-system-being-built will provide tutoring. In other words, it simply represents ways in which the given problem can be solved. CTAT provides a tool, the Behavior Recorder, for creating them easily. Behavior graphs help in analyzing the knowledge needs, support thinking about transfer, and thereby guide the development of a cognitive model.

As a representation of the solution space of a problem, behavior graphs are not tied to any particular type of tutor and are likely be useful across a range of tutor authoring tools, especially those addressing tutoring for problems with a more complex solution space. For example, they may be helpful in tools for building constraint-based tutors. They may be less useful in the ASSISTments tool, given ASSISTments

strongly constrains the variability of the problems' solution space, with each problem essentially having one single-step path and multi-step path, the latter representing the scaffolded, version.

As the CTAT author creates a behavior graph, an xPST author begins to construct the task sequence and goal-nodes in xPST pseudo-code. In both cases, these authoring steps are a reflection of the tasks and knowledge components that the author is indicating as needed in order for a learner to do the task. ASPIRE has the author identify those tasks upfront, before the author creates the examples, based on the ontology that the author creates. SimStudent's induction of the task's rules depends on the representation being used, so the author's CTA is important in shaping what the learned rules will look like.

After the author has created the first version of the tutor and students have gone through its instruction, some of these systems have features that enable the authors to iterate the design of the tutor using student log files to inform a CTA and a redo of the tutor. ASSISTments, CTAT, and SimStudent all have robust ways for researchers and teachers to examine learner responses and adapt their tutor's instruction accordingly. ASSISTments produces a report showing learners' most common wrong answers, and also allows students to comment on the problems. SimStudent has a tool to validate its cognitive model by model-tracing through student log data to ensure correct functioning. Initial work has shown that this improves the quality of the model, though additional work will have to be performed to see how much it improves student learning.

Recommendations for GIFT

Having reviewed the challenges and benefits of example-based tutor authoring, we offer suggested features for GIFT so that it may also benefit from this approach. We begin with a brief summary of its architecture from the authoring perspective. GIFT is closer to ASPIRE than the other tools, in that its tutors can be viewed as a collection of states to be reached or constraints to be satisfied, without a particular procedural order to be followed. While sequencing can be achieved through conditions and subconditions, GIFT's core is designed around states. In particular, in GIFT's domain knowledge file (DKF) editor, typically used for authoring, there are tasks, which have concepts, which, in turn, have conditions, which, in turn, trigger feedback (Figure 10). The tasks are collections of states to be achieved. The concepts (with possible subconcepts) are analogous to learner skills used. Concepts are designed as learned if their conditions and subconditions are met. There is not a specific analogy to a procedural step that a learner might take, but a DKF condition is similar. If a step is taken, a condition is likely met. Note that the feedback assigned to be given when a condition is met is chosen from a menu of possible feedback items. Thus, a given feedback item can be reused easily by the author in multiple conditions.

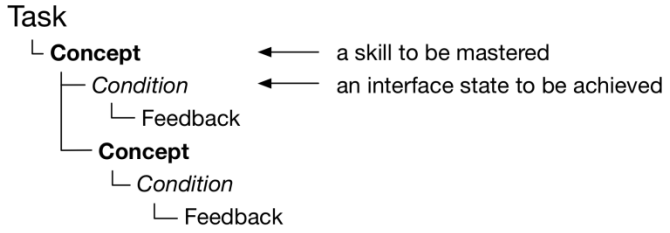

Figure 10: GIFT's DKF format, typically used for authoring

Conditions might be based on whether a certain time has passed in the simulation, or whether the learner has reached a specific location or state. At this early point within GIFT's development, there is not any way to combine conditions in its DKF authoring module, e.g., if the learner does X (Condition 1) while

also in location Y (Condition 2), then perform a particular action. However, it does have an additional authoring tool, SIMILE, that works more like a scripting engine in which authors write explicit if…then code, in which this is possible.

In GIFT there is not a natural way to represent a procedural solution path or branching at a decision point such as in CTAT or xPST. Software applications that manage the passage of time (e.g., video editing suites or medical systems monitoring patient data), aka "timeline-navigators" (Rubio, 2014), typically have a timeline and playhead metaphor as part of their user interface. An analogous interface is recommended for GIFT to indicate to the author the current status of the internal condition evaluations, though it would not likely map cleanly onto a linear timeline, since GIFT looks for active concepts and then evaluates their conditions. Whenever conditions are true, they generate feedback, which may accumulate across multiple conditions. While it is feasible with significant management of the conditions to create a sequence with branching points, the underlying architecture does not make this a natural task for an author. Also, GIFT does not differentiate between forms of feedback, such as hints, prompts, or buggy messages based on incorrect answers.

This state-based and less procedural approach makes GIFT much better adapted to tutors on simulations that enable multiple complex states, such as game engines. A 3D game engine scenario, with multiple live player entities and some game-based non-player characters, is difficult to frame as a procedural tutor and is better approached as a network of noteworthy states (Devasani, Gilbert, Shetty, Ramaswamy & Blessing, 2011; Gilbert, Devasani, Kodavali & Blessing, 2011; Sottliare & Gilbert, 2011). Game engines often have level editors that allow almost WYSIWYG editing and scripting by non-programmers. These could be an inspiration for GIFT. However, since GIFT is essentially an abstraction layer used to describe conditions and states within such a system, enabling the author to visualize the learner's experience within the simulation while simultaneously understanding the current state of the tutor is a complex challenge for which that are not many common user interface precedents. Currently within GIFT, it is difficult to preview and debug the learner's experience using the tutor or to easily encode a particular example into GIFT. The CTA of a tutoring experience (described above) must first be created separately and then be transformed to match GIFT's state-based condition architecture. Once authored, this architecture also makes it difficult to conduct quality assurance testing. The condition-based tutor can be complex to test because the author must think through all possible combinations of states that might generate feedback.

In terms of example-based authoring, a given GIFT tutor is essentially one large example; there is no particular mechanism for generalization. However, GIFT is highly modular, so that elements of a given tutor such as the feedback items can be re-used in other tutors. The features of the aforementioned tutoring systems that promote generalization of rules and easy visualization of the learner's experience via the authoring tool would be ones for GIFT to emulate.

References

Adams, D. M., McLaren, B. M., Durkin, K., Mayer, R. E., Rittle-Johnson, B., Isotani, S. & Velsen, M. V. (2014). Using erroneous examples to improve mathematics learning with a web-based tutoring system. *Computers in Human Behavior, 36,* 401 - 411. doi:10.1016/j.chb.2014.03.053}

Aleven, V. (2010). Rule-Based cognitive modeling for intelligent tutoring systems. In R. Nkambou, J. Bourdeau & R. Mizoguchi (Eds.), Studies in Computational Intelligence: Vol. 308. *Advances in intelligent tutoring systems* (pp. 33-62). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-14363-2_3

Aleven, V. & Koedinger, K. R. (2013). Knowledge component approaches to learner modeling. In R. Sottilare, A. Graesser, X. Hu & H. Holden (Eds.), *Design recommendations for adaptive intelligent tutoring systems* (Vol. I, Learner Modeling, pp. 165-182). Orlando, FL: US Army Research Laboratory.

Aleven, V., McLaren, B. M. & Sewall, J. (2009). Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE Transactions on Learning Technologies, 2*(2), 64-78

Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley & T. W. Chan (Eds.), *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 61-70). Berlin: Springer Verlag.

Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2009). A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *International Journal of Artificial Intelligence in Education, 19*(2), 105-154.

Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S. & Koedinger, K. R. (under review). Toward tutoring at scale: Reflections on "A new paradigm for intelligent tutoring systems: Example-tracing tutors." Submitted to the *International Journal of Artificial Intelligence in Education.*

Aleven, V., Sewall, J., McLaren, B. M. & Koedinger, K. R. (2006). Rapid authoring of intelligent tutors for real-world and experimental use. In Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson & W. Didderen (Eds.), *Proceedings of the 6th IEEE international conference on advanced learning technologies (ICALT 2006)* (pp. 847-851). Los Alamitos, CA: IEEE Computer Society

Anderson, J. R. & Pelletier, R. (1991). A development system for model-tracing tutors. In *Proceedings of the International Conference of the Learning Sciences*, 1-8. Evanston, IL.

Anderson, J. R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences, 4*(2), 167-207.

Baker, R. S. J. d., Corbett, A. T. & Koedinger, K. R. (2007). The difficulty factors approach to the design of lessons in intelligent tutor curricula. *International Journal of Artificial Intelligence and Education, 17*(4), 341-369.

Blessing, S. B., Devasani, S. & Gilbert, S. (2011). Evaluation of webxpst: A browser-based authoring tool for problem-specific tutors. In G. Biswas, S. Bull & J. Kay (Eds.), *Proceedings of the Fifteenth International Artificial Intelligence in Education Conference* (pp. 423-425), Auckland, NZ. Berlin, Germany: Springer.

Blessing, S. B., Gilbert, S., Ourada, S. & Ritter, S. (2009). Authoring model-tracing cognitive tutors. *International Journal for Artificial Intelligence in Education, 19*, 189-210.

Clark, R. E. & Estes, F. (1996). Cognitive task analysis. *International Journal of Educational Research*. 25(5). 403-417.

Clark, R. E., Feldon, D., van Merriënboer, J., Yates, K. & Early, S. (2007). Cognitive task analysis. In J. M. Spector, M. D. Merrill, J. J. G. van Merriënboer & M. P. Driscoll (Eds.), *Handbook of research on educational communications and technology (3rd ed.).* (pp. 577-93). Mahwah, NJ: Lawrence Erlbaum Associates.

Corbett, A. T. & Anderson, J. R. (1995). Knowledge decomposition and subgoal reification in the ACT programming tutor. *Artificial Intelligence and Education*, 1995: The Proceedings of AI-ED 95. Charlottesville, VA: AACE.

Corbett, A., Kauffman, L., MacLaren, B., Wagner, A. & Jones, E. (2010). A cognitive tutor for genetics problem solving: Learning gains and student modeling. *Journal of Educational Computing Research, 42*(2), 219-239.

Cypher, A. (Ed.). (1993). *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.

Devasani, S., Gilbert, S. & Blessing, S. B. (2012). Evaluation of two intelligent tutoring system authoring tool paradigms: Graphical user interface-based and text-based. *Proceedings of the 21st Conference on Behavior Representation in Modeling and Simulation* (pp. 54-61), Amelia Island, FL.

Devasani, S., Gilbert, S. B., Shetty, S., Ramaswamy, N. & Blessing, S. (2011). Authoring Intelligent Tutoring Systems for 3D Game Environments. *Presentation at the Authoring Simulation and Game-based Intelligent Tutoring Workshop at the Fifteenth Conference on Artificial Intelligence in Education*, Auckland.

Gilbert, S. B., Blessing, S. B. & Blankenship, E. (2009). The accidental tutor: Overlaying an intelligent tutor on an existing user interface. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*.

Gilbert, S., Devasani, S., Kodavali, S. & Blessing, S. B. (2011). Easy authoring of intelligent tutoring systems for synthetic environments. *Proceedings of the 20th Conference on Behavior Representation in Modeling and Simulation* (pp. 192-199), Sundance, UT.

Green, T. R. G. & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, *7*, 131- 174.

Jarvis, M. P., Nuzzo-Jones, G. & Heffernan, N. T. (2004). Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. In J. C. Lester (Ed.), *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 541-553). Heidelberg, Berlin: Springer.

Koedinger, K. R. & Nathan, M. J. (2004). The real story behind story problems: Effects of representations on quantitative reasoning. *The Journal of the Learning Sciences*, *13*(2), 129-164.

Koedinger, K. R., Aleven, V. & Heffernan, N. (2003). Toward a rapid development environment for cognitive tutors. In U. Hoppe, F. Verdejo & J. Kay (Eds.), *Proceedings of the International Conference on Artificial Intelligence in Education* (pp. 455-457). Amsterdam: IOS Press

Koedinger, K. R., Anderson, J. R., Hadley, W. H. & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8*, 30-43.

Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B. & Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In J. C. Lester, R. M. Vicario & F. Paraguaçu (Eds.), *Proceedings of seventh international conference on intelligent tutoring systems, ITS 2004* (pp. 162-174). Berlin: Springer.

Lau, T. A. & Weld, D. S. (1998). Programming by demonstration: An inductive learning formulation *Proceedings of the 4th international conference on Intelligent user interfaces* (pp. 145-152). New York, NY: ACM Press

Li, N., Matsuda, N., Cohen, W. W. & Koedinger, K. R. (2015). Integrating Representation Learning and Skill Learning in a Human-Like Intelligent Agent. *Artificial Intelligence*, 219, 67-91.

Lieberman, H. (2001). *Your wish is my command: Programming by example*. San Francisco, CA: Morgan Kaufmann.

Long, Y. & Aleven, V. (2013). Supporting students' self-regulated learning with an open learner model in a linear equation tutor. In H. C. Lane, K. Yacef, J. Mostow & P. Pavlik (Eds.), *Proceedings of the 16th international conference on artificial intelligence in education (AIED 2013)* (pp. 249-258). Berlin: Springer

Lovett, M. C. (1998). Cognitive task analysis in service of intelligent tutoring system design: a case study in statistics. In B. P. Goettl, H. M. Halff, C. L. Redfield & V. Shute (Eds.) *Intelligent Tutoring Systems, Proceedings of the Fourth International Conference* (pp. 234-243). Lecture Notes in Computer Science, 1452. Berlin: Springer-Verlag.

Lovett, M., Meyer, O. & Thille, C. (2008). JIME-The open learning initiative: Measuring the effectiveness of the OLI statistics course in accelerating student learning. Journal of Interactive Media in Education, 2008(1).

Maass, J. K. & Blessing, S. B. (April, 2011). Xstat: An intelligent homework helper for students. Poster presented at the 2011 Georgia Undergraduate Research in Psychology Conference, Kennesaw, GA.

MacLellan, C., Koedinger, R. K. & Matsuda, N. (2014). Authoring tutors with SimStudent: An evaluation of efficiency and model quality. In S. Trausen-Matu & K. Boyer (Eds.), *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 551-560). Switzerland: Springer.

Martin, B. & Mitrovic, A. (2002). WETAS: a web-based authoring system for constraint-based ITS. In: P. de Bra, P. Brusilovsky and R. Conejo (eds) *Proc. 2nd Int. Conf on Adaptive Hypermedia and Adaptive Web-based Systems AH 2002*, Malaga, Spain, LCNS 2347, 543-546.

Matsuda, N., Cohen, W. W. & Koedinger, K. R. (2005). Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors *AAAI Workshop on Human Comprehensible Machine Learning (Technical Report WS-05-04)* (pp. 1-8). Menlo Park, CA: AAAI association.

Matsuda, N., Cohen, W. W. & Koedinger, K. R. (in press). Teaching the Teacher: Tutoring SimStudent leads to more Effective Cognitive Tutor Authoring. *International Journal of Artificial Intelligence in Education*.

Matsuda, N., Lee, A., Cohen, W. W. & Koedinger, K. R. (2009). A computational model of how learner errors arise from weak prior knowledge. In N. Taatgen & H. van Rijn (Eds.), *Proceedings of the Annual Conference of the Cognitive Science Society* (pp. 1288-1293). Austin, TX: Cognitive Science Society.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2008). Why tutored problem solving may be better than example study: Theoretical implications from a simulated-student study. In B. P. Woolf, E. Aimeur, R. Nkambou & S. Lajoie (Eds.), *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 111-121). Heidelberg, Berlin: Springer.

McLaren, B. M., Adams, D., Durkin, K., Goguadze, G., Mayer, R. E., Rittle-Johnson, B., . . . Velsen, M. V. (2012). To err is human, to explain and correct is divine: A study of interactive erroneous examples with middle school math students. In A. Ravenscroft, S. Lindstaedt, C. Delgado Kloos & D. Hernández-Leo (Eds.), *21st century Learning for 21st Century Skills:7th European Conference of Technology Enhanced Learning, EC-TEL 2012* (pp. 222-235). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-33263-0_18

McLaren, B. M., DeLeeuw, K. E. & Mayer, R. E. (2011a). Polite web-based intelligent tutors: Can they improve learning in classrooms? *Computers & Education, 56*(3), 574-584.

McLaren, B. M., DeLeeuw, K. E. & Mayer, R. E. (2011b). A politeness effect in learning with web-based intelligent tutors. International Journal of Human Computer Studies, 69(1-2), 70-79. doi:10.1016/j.ijhcs.2010.09.001

Means, B. & Gott, S. (1988). Cognitive task analysis as a basis for tutor development: Articulating abstract knowledge representations. In J. Pstotka, L.D. Massey & S.A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (pp.35-57). Hillsdale, NJ: Lawrence Erlbaum Associates.

Mitrovic, A. (1998). Experiences in implementing constraint-based modelling in SQL-tutor. In Goettl, B.P., Halff, H.M., Redfield, C.L. and Shute, V.J. (Eds.), *Proceedings of Intelligent Tutoring Systems*, 414-423.

Mitrovic, A. (2012). Fifteen years of constraint-based tutors: What we have achieved and where we are going. *User Modeling and User-Adapted Interaction*, *22*, 39-72.

Mitrovic, A. & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language, *International Journal of Artificial Intelligence in Education*, *10*, 238-256.

Mitrovic, A., Koedinger, K. R. & Martin, B. (2003). A Comparative analysis of cognitive tutoring and constraint-based modeling. In: Brusilovsky, P., Corbett, A., and de Rosis, F. (Eds.) *Proceedings of User Modelling*, 313-322.

Mitrovic, A., Martin, B. & Suraweera, P. (2007). Intelligent tutors for all: Constraint-based modeling methodology, systems and authoring. *IEEE Intelligent Systems*, *22*, 38-45.

Mitrovic, A., Williamson, C., Bebbington, A., Mathews, M., Suraweera, P., Martin, B., Thomson, D. & Holland, J. (2011). An Intelligent Tutoring System for Thermodynamics. *EDUCON 2011*, Amman, Jordan, 378-385.

Muggleton, S. & de Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19-20*(Supplement 1), 629-679.

Nardi, B. A. (1993). *A small matter of programming: Perspectives on end-user computing*. Boston, MA: MIT press.

Newell, A. & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Ohlsson, S. (1994). Constraint-based student modelling, in *Student modelling: The key to individualized knowledge-based instruction*, 167-189.

Ohlsson, S. (1996). Learning from performance errors. *Psychological Review*, *103*, 241-262.

Ohlsson, S. & Mitrovic, A. (2007). Fidelity and efficiency of knowledge representations for intelligent tutoring systems. *Technology, Instruction, Cognition and Learning*, *5*, 101-132.

Olsen, J. K., Belenky, D. M., Aleven, V. & Rummel, N. (2014). Using an intelligent tutoring system to support collaborative as well as individual learning. In S. Trausan-Matu, K. E. Boyer, M. Crosby & K. Panourgia (Eds.), *Proceedings of the 12th International Conference on Intelligent Tutoring Systems, ITS 2014* (pp. 134-143). Berlin: Springer. doi:10.1007/978-3-319-07221-0_66

Olsen, J. K., Belenky, D. M., Aleven, V., Rummel, N., Sewall, J. & Ringenberg, M. (2014). Authoring tools for collaborative intelligent tutoring system environments. In S. Trausan-Matu, K. E. Boyer, M. Crosby & K. Panourgia (Eds.), *Proceedings of the 12th International Conference on Intelligent Tutoring Systems, ITS 2014* (pp. 523-528). Berlin: Springer. doi:10.1007/978-3-319-07221-0_66

Ostrow, K. & Heffernan, N. T. (2014). Testing the multimedia principle in the real world: a comparison of video vs. Text feedback in authentic middle school math assignments. In *Proceedings of the 7th international conference on educational data mining* (pp. 296-299).

Rau, M. A., Aleven, V. & Rummel, N. (2015). Successful learning with multiple graphical representations and self-explanation prompts. *Journal of Educational Psychology, 107*(1), 30-46. doi:10.1037/a0037211

Rau, M. A., Aleven, V., Rummel, N. & Pardos, Z. (2014). How should intelligent tutoring systems sequence multiple graphical representations of fractions? A multi-methods study. *International Journal of Artificial Intelligence in Education, 24*(2), 125-161.

Rau, M. A., Aleven, V., Rummel, N. & Rohrbach, S. (2013). Why interactive learning environments can have it all: Resolving design conflicts between conflicting goals. In W. E. Mackay, S. Brewster & S. Bødker (Eds.), *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2013)* (pp. 109-118). ACM, New York.

Razzaq, L. M. & Heffernan, N. T. (2009, July). To tutor or not to tutor: That is the question. In *AIED* (pp. 457-464).

Razzaq, L., Patvarczki, J., Almeida, S. F., Vartak, M., Feng, M., Heffernan, N. T. & Koedinger, K. R. (2009). The assistment builder: Supporting the life cycle of tutoring system content creation. *Learning Technologies, IEEE Transactions on*, *2*(2), 157-166.

Reed, S. K. & Bolstad, C. A. (1991). Use of examples and procedures in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*, 753-766.

Ritter, S. & Koedinger, K. R. (1996). An architecture for plug-in tutor agents. *International Journal of Artificial Intelligence in Education*, *7*, 315-347.

Roll, I., Holmes, N. G., Day, J. & Bonn, D. (2012). Evaluating metacognitive scaffolding in guided invention activities. *Instructional Science, 40*(4), 1-20. doi:10.1007/s11251-012-9208-7

Rubio, E. (2014) Defining a software genre: Timeline navigators. (Unpublished Master's thesis). Iowa State University, Ames, IA.

Sottilare, R. and Gilbert, S. B. (2011). Considerations for adaptive tutoring within serious games: authoring cognitive models and game interfaces. *Presentation at the Authoring Simulation and Game-based Intelligent Tutoring Workshop at the Fifteenth Conference on Artificial Intelligence in Education*, Auckland.

Stampfer, E. & Koedinger, K. R. (2013). When seeing isn't believing: Influences of prior conceptions and misconceptions. In M. Knauff, M. Pauen, N. Sebanz & I. Wachsmuth (Eds.), *Proceedings of the 35th Annual Conference of the Cognitive Science Society* (pp. 916-919). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-39112-5_145

Suraweera, P., Mitrovic, A. & Martin, B. (2010). Widening the knowledge acquisition bottleneck for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, *20*(2), 137-173.

Suraweera, P., Mitrovic, A., Martin, B., Holland, J., Milik, N., Zakharov, K. & McGuigan, N. (2009). Ontologies for authoring instructional systems. D. Dicheva, R. Mizoguchi, J. Greer (eds.) *Semantic Web Technologies for e-Learning*. IOS Press, (pp. 77-95).

VanLehn , K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, *16*(3), 227-265.

Waalkens, M., Aleven, V. & Taatgen, N. (2013). Does supporting multiple student strategies lead to greater learning and motivation? Investigating a source of complexity in the architecture of intelligent tutoring systems. *Computers & Education*, *60*(1), 159-171.

Westerfield, G., Mitrovic, A. & Billinghurst, M. (2013). Intelligent augmented reality training for assembly tasks. In: H. C. Lane, K. Yacef, J. Mostow, O. Pavlik (Eds.): *Proceedings of the Sixteenth International Conference of Artificial Intelligence in Education*, LNAI 7926, pp. 542-551. Springer, Heidelberg.

Wylie, R., Sheng, M., Mitamura, T. & Koedinger, K. (2011). Effects of adaptive prompted self-explanation on robust learning of second language grammar. In G. Biswas, S. Bull, J. Kay & A. Mitrovic (Eds.), *Proceedings of the 15th International Conference on Artificial Intelligence in Education, AIED 2011* (pp. 588-590). Springer Berlin Heidelberg. doi:10.1007/978-3-642-21869-9_110